IA317 Challenge report

WANG Yuqing
ZHU Fangda

This challenge is to predict a device's location (latitude and longitude). There are 13331410 samples in the training set. Each sample contains those information:

```
data_train2.columns
```

```
Index(['messageid', 'latitude', 'longitude', 'did', 'did_hex', 'time_msg',
       'time_ux', 'time_ux_client', 'motion', 'speed', 'altitude',
       'nb_satellites', 'data_type', 'radius', 'datepart', 'seqnumber', 'dtid',
       'nseq', 'rssi', 'bsid_hex', 'bsid', 'snr', 'freq'],
      dtype='object')
```

The 'latitude' and 'longitude' are the information that we need to predict. Each 'messageid' represent to a message, send by a device. Then, this message was received by one or several base station. The datamart_production_basestations_history.csv contains the information of base stations. Before getting started, I find the latitudes and longitudes of base stations, and I stored them in the bs.csv.

First I tried to analyse what are the useful features in the training set. I re-grouped training set. The items with the same message_id will be put together. I found out that: for a message which is received by several base stations, their 'rssi', 'snr', 'nseq' information are different. The rest of the information are all the same. So, I think we can't predict the device's location by using the time difference (which meas the the time difference between receive time and emission time). We can only use 'rssi' information to calculate the distance between the device and the base station, and then use this distance together with the base stations' location to roughly calculate the devices' location.

So, for each sample in the training data, we need to calculate the distance between the device and the base station which received this message. It's a problem of calculating in the spherical coordinate system.

```
@njit
def calculate_each_distance(y,bs_index, base):
    distance = np.zeros(len(y))

    for i in range(len(y)):
        latA = (y[i][0] * math.pi/180.0)
        lngA = (y[i][1] * math.pi/180.0)
        index = bs_index[i]
        latB = (base[index][0] * math.pi/180.0)
        lngB = (base[index][1] * math.pi/180.0)
        a = latA - latB
        b = lngA - lngB
        s = 2*6371*math.asin(math.sqrt(math.pow(math.sin(a/2),2)
                                       +math.cos(latA)*math.cos(latB)*math.pow(math.sin(b/2),2)))
        distance[i] = s
    return distance
```

Here, in order to accelerate the calculation, I imported numba, and I add @njit as decorator. The whole calculating part would be very fast. After that, I created a new column 'distance' and put it in the training data. After that, I also calculated and stored the angle between the device

and base station (Although it doesn't seem to have relation with other features in the training set).

The principle of the 'rssi distance measurement' is:

$$P_r - A = -10n\log(r)$$

In this formula, P_r is the received signal strength indicator (rssi), r is the distance, A and n are some parameters. Here, I took A as 47, n as 3.875. And I calculate this r roughly for both data set and stored it.

```
data_train['rssi2'] = 10 ** ((np.abs(data_train['rssi'])-47)/38.75)
data_validation['rssi2'] = 10 ** ((np.abs(data_validation['rssi'])-47)/38.75)
```

Then, I thought that maybe the data_type and dtid could be useful since the transmit signal strength and the signal type are also important in calculating distant. Different kind of device may transmit different strength of signal. Those two string are with the format of 'string', so I used the OneHotEncoder to transform them.

The next step is to set X_train, X_test, y_train. I took only the 'rssi', 'rssi2' (I created before), 'snr', 'data_type', 'dtid' (with onehot encoder). And use them to form the X_train and X_test. y_train is the 'distance' column in the training data set.

After that, it's time to find a proper regression method and set parameters. I tried linear regression, random forest and the xgboost. The random forest(n_estimators = 100, max_depth=10) and the xgboost(with the parameters: n_estimators = 100, learning_rate = 0.08) would take nearly one hour to fit the train and test. On the contrary, the linear regression can fit quickly can has even a better performance on the the predict result. So finally I chose to apply linear regression model.

So by using the predict function in the linear model, I got the prediction of the distance between device and base station. The next step is to group the test data by the message_id. The number of the base station is 159, so there will be 159 columns in the new data set. Each column correspond to the distance between a certain base station and the device. However, this new data set contains so many NaN. In fact, for most of the message_id, there are usually only 1 or 2 base station who received the message. The regression can handle a data set with NaN, and I don't think we can replace those NaN by 0.

```
g = data_validation.groupby('messageid').groups
dict_index={}
message_id = []
array_distance = scipy.sparse.lil_matrix((len(g), len(list_bsid)))
j = 0
for gg in g:
    temp = []
    message_id.append(data_validation['messageid'][g[gg][0]])
    for i in range(len(g[gg])):
        message_index = g[gg][i]
        temp.append(g[gg][i])
        bsid_index = list_bsid.index(data_validation['bsid'][message_index])
        message_distance = y_test['distance'][message_index]
        array_distance[j, bsid_index] = message_distance
    dict_index[j] = temp
    j += 1
```

So the final method I chose is using base stations' location to help to get the result. Before that, I also tried to predict the angle between the base station and the device. But finally, I couldn't find a good model to predict it. At last, begin to calculate the location of device. As for the message which is only received by one base station, I don't have method but can only set the location of this base station as the location. As for the message which is received by 2 base stations, I find a coordinate between those two base stations by use the proportion of the predicted distance to those two stations. As for the message which is received by more than 2 base stations, I'm not sure how to deal it, and I just used a rough way to calculate them. Then I stored them as my y_test.

```
def calculate_lat_lng(X, bs):
    y = np.zeros((len(X),2))
    for i in range(len(X)):
        num_point = 0
        point_distance = []
        point_index = []
        for j in range(159):
            if X[i,j] != 0:
                point_distance.append(X[i,j])
                point_index.append(j)
                num_point += 1
        if num_point == 1:
            index = int(point_index[0])
            y[i][0] = bs[index,0]
            y[i][1] = bs[index,1]
        if num_point == 2:
            index1 = int(point_index[0])
            index2 = int(point_index[1])
            y[i][0] = ((bs[index1,0]*point_distance[1])+(bs[index2,0]*point_distance[0]))
                        /(point_distance[1]+point_distance[0])
            y[i][1] = ((bs[index1,1]*point_distance[1])+(bs[index2,1]*point_distance[0]))
                        /(point_distance[1]+point_distance[0])
        if num_point > 2:
            point_distance = np.array(point_distance)**2
            sum_distance = sum(point_distance)
            for k in range(num_point):
                index = int(point_index[k])
                y[i][0] = y[i][0] + (bs[index,0]*point_distance[k])/sum_distance
                y[i][1] = y[i][1] + (bs[index,1]*point_distance[k])/sum_distance
    return y
```

Finally the score of my result is 8.86 km.

Something that may improve the result:

1. Make good use of 'motion', 'speed', 'radius' features (which I didn't try because of the time limit).
2. For the same message and the same base station, the 'rssi' and the 'snr' would be different. Maybe we should deal with them before throwing them in to the regression model.
3. There could be a better model that I haven't try and could predict distance more precise.
4. Maybe the angle between the base station and the device could be predicted by the data set we got.
5. After getting the predicted distance, will there be a model which can deal with matrix of 159 columns which contains many NaN?
6. If not, in my may (the calculate_lat_lng() I wrote), how can I get a better way to calculate the location of the device whose message was received by more the 2 base stations?